Programming Basics in Objective-C

Objective-C is a very elegant language. It mixes the efficiency of the C language with the object-oriented goodness of Smalltalk. This combination was introduced in the mid-1980s and is still powering the fantastic applications behind the iPhone, iPad, and Mac OS X. How does a language that is over 20 years old stay relevant and useful after all of that time? Well, some of its success has to do with the fact that the two languages that make up Objective-C are very well tested and very well designed. Another reason is less obvious; the various frameworks available for the iPhone and Mac OS X make developing full-featured applications much easier. These frameworks benefit from the fact that they have been around awhile, which equates to stability and high functionality. Lastly, Objective-C is highly dynamic. While we won’t be focusing on that particular feature in this chapter, the dynamic nature of Objective-C provides a flexibility not found in many compiled languages. With all of these great features, Objective-C and the corresponding frameworks provide an excellent palette from which to create a masterpiece!

This chapter will introduce some of the more common concepts of Objective-C, such as properties and collection classes. This chapter will also show how properties and instance variables are used from within Xcode when dealing with user-interface elements. This sounds like a lot to accomplish, but Objective-C, the Foundation framework, and the Xcode tool provide a wealth of objects and methods and a way to build applications with ease.

Collections

Understanding collections is a fundamental part of learning Objective-C. In fact, collection objects are fundamental constructs of nearly every modern object-oriented language library—sometimes they are referred to as containers. Simply put, a collection is a type of class that can hold and manage other objects. The whole purpose of a collection is that it provides a common way to store and retrieve objects efficiently.
There are several different types of collections. While they all fulfill the same purpose of being able to hold other objects, they differ mostly in the way objects are retrieved. Here is a list of the most common collections used in Objective-C:

- NSSet
- NSArray
- NSDictionary
- NSMutableSet
- NSMutableArray
- NSMutableDictionary

Notice that, among the three collection classes listed, there is one that contains the word `Mutable`. A mutable (vs. non-mutable) class is one that can have items added and removed from it after the collection has been created. Mutable means that it can be modified. Non-mutable collections must be initialized with all of their values when they are first created and, thereafter, cannot be modified.

### Using NSSet

The NSSet class is used to store an unordered list of objects. *Unordered* means exactly that—the objects are stored in the set without regard to order. The advantage of the NSSet is performance—it’s the fastest collection class available. Use NSSet when it is necessary to store a collection of objects and the order in which they are stored or retrieved is not crucial.

Here is a typical NSSet initialization method:

```objc
NSSet *mySet = [NSSet setWithObjects:@"String 1", @"String 2", @"Whatever", nil];
```

As you can see, the set is initialized with a list of objects, in this case a list of strings. The last object must be `nil` to indicate the end of the list of objects. Also, the example uses strings, but an NSSet can be comprised of any object or even different types of objects, including objects from other collections!

**TIP:** All collection classes have the ability to store and manage any type of object at once. However, in typical cases, most programmers tend to store a single type of object in any one particular collection class to make the code less complicated.

In order to go after data in the NSSet, a few typical methods of accessing the elements within an NSSet are used. One method, as shown in Listing 8–1, is to use what is referred to as a *fast enumerator* and retrieve each object one by one. Note that the fast enumeration below (lines #3–5) works on all collection classes.